

Portable Program Review

Vol. 2, No. 2

February, 1986

THE MISSING LINK

In this subroutine, David C. Hart of South Yarmouth, Massachusetts provides the tools for making programs more professional-looking and, yes, user-friendly. -Ed.

Have you ever wished for a solid foundation upon which to build your next program? You know that something's keeping your work from being the masterpiece that it ought to be -- that little extra that makes it something special.

Perhaps what you need is a comprehensive, easy to use screen input subroutine that combines most of the features of TEXT with special options to provide that professional feel to your program. I'm offering one general-purpose data entry routine, MISLIN.BA, which has the following features:

1. Easily formatted inputs of 1 to 239 characters, fitting into six lines with one space left for cursor overflow.

2. Handles INPUT\$(1) entries as well as alphanumeric strings.
3. Depicts the assigned field using underscore characters, making your programs easier to use for both novices and experts. Who can remember how long a field is allowed to be at each input line?
4. Allows use of the top line for headings, the label line for prompts, and the remaining six in between for query prompts, display and input strings -- without screen scrolling.
5. Has full cursor control throughout the maximum six line field, as in TEXT mode.
6. Provides for relational checking of the input string before returning from the routine to shorten the logical expression coding.
7. Produces audible alerts, but only for the most important conditions.

Details, Details

While the subroutine's size may qualify it as a full fledged program, don't let the length scare you. I originally developed it for use in an invoicing program, so you'll find it as adaptable for number juggling as string manipulation.

Typing speed is tied directly to the amount of free string

IN THIS ISSUE:

THE MISSING LINK	1
KEEPING IT CLEAN	6
INTERNAL AUDIT	8

space, so CLEAR enough space in your initialization line (2,000 bytes is fine for most applications).

Lines 5 thru 8 may be used alone in some short programs needing only single keystroke entries.

The routine operates from within the program in the same manner as TEXT, with only a few exceptions:

1. Block move functions (F5, F6, F7, PASTE), the find function (F1), and word wrap are not included. The first two wouldn't be worth the memory, since the largest allowable input is 239 characters.
2. As presented, the tab function inserts four spaces instead of TEXT's eight, though you can change this. Instead of inserting a control code, printable spaces (CHR\$(32)) are inserted. To delete tabs, many backspaces have to be entered. To alter the number of spaces, change O3=4 in line 1070 to your new choice and the five in the same line to your new value plus 1.
3. The Escape key will erase everything from the current cursor position to the end of the displayed alphanumeric string. The resulting void is filled with underscore characters. Be careful when using the Escape key since the erased information is irretrievable. It's very handy for clearing whole screens or deleting default answers to questions: The user can hit Enter if the suggestion is correct or Escape if it's not.
4. Since during a Shift-Right or Shift-Left arrow operation the routine searches for the first character following a space, it

will find the beginning of the next word only if the characters are separated by a single space. Should it find a succession of spaces, the arrows will take you forward or backward one space at a time until you pass through them.

Pick A Variable, Any Variable

As you can see, there are no major deviations from the TEXT format, so keyboard use of the routine should be familiar.

Following is a list of the variables used in the routine. The starred ones should not be used elsewhere in the applications program. You'll find that none carry type declaration tags, since they consume one byte each and the routine would need a lot of them. The variable types are listed first, and you should include them in the DEF statements in your program's initialization line.

Should your numeric data require 14-digit accuracy, don't define variable E. It will then be double precision by default.

A note here for the beginning programmer: Remember when defining your variables that BASIC defaults to double precision all that you leave unnamed. Therefore, define as integers everything you don't have plans for immediately and change them as needed during the programming process. Like everything else, numbers occupy a predetermined amount of space in memory. Integers take 2 bytes, single precision take 4, and double precision take 8 bytes each. Don't let the name "double precision" fool you. They're not necessarily twice as accurate as single precision numbers within your application. Since BASIC does all of its calculations in double precision accuracy, your type designation only determines how your answer will be rounded off.

Variable types

Integer - G, O
Single precision - E
String - F, K, N

Variable listing

K = compiled string (returned as input string)

K0 = character input from keyboard

N = string sent out for editing or suggested input and converted to K

F0 = working variable

FQ = string sent as query prompt or screen information

G1 = PRINT@ position for FQ

G2 = PRINT@ position for N

G3 = assigned maximum input string length

G4 = tracks cursor position

G5 = working variable

G, O and O1-09 = counters and flags. O1-09, G0 and E0 are relational checks at the subroutine's end.

E0 = numeric value of returned string K -- change precision to meet your requirements.

Special Cases

There are five variables which must be set before a GOSUB 1000 and three for a GOSUB 5.

G1 - For both GOSUB 1000 and GOSUB 5, set to the PRINT @ position for the query prompt or display string (FQ).

G2 - For both subroutines, set to the PRINT @ position for the input string or string to be

edited (N). This value must be greater than 0.

G3 - For a GOSUB 1000 only, set to the maximum allowable length of the input string (1 - 239). This will show as a field of underscore characters and an audible warning will sound if you attempt to exceed its limits.

N - For a GOSUB 1000 only, place the string you wish to edit or the suggested input into this. It will be displayed with underscore characters filling the unused portion of the assigned field. Should you use a suggested input, you can erase it by hitting the Escape key before typing in a different answer.

FQ - For both subroutine calls, place the query prompt or any other screen display variable into variable. Should you have use for more than one in a special application, simply choose another variable name and insert a PRINT @ command at the beginning of line 1010. You must also choose a PRINT @ position variable which will not conflict with one in the routine.

Return Codes

The string returned from the routine (K) is filtered through a set of relational expressions which make the logical expressions much shorter and easier to code. These relational expressions may be expanded upon or altered as your requirements change.

Following is a list of the ones I used. You'll notice that the alpha characters are checked in both upper and lower case. Following each are the most common uses I have for them. All except O2 and E0 are returned as -1 for true and 0 for false.

00 = D or d (done)
01 = E or e (exit)
02 = returned string length
03 = Y or y (yes)
04 = N or n (no)
05 = F or f (forward)
06 = NOT Y or y or N or n (not
yes or no)
07 = B or b (backward)
08 = numeric value is 0 ?
09 = returned string null ?
E0 = numeric value of returned
string
G0 = C or c (change)

GOSUB 5 is for aimed at single-keystroke entries in answer to menus. GOSUB 5 is simply an INPUT\$(1) statement, accepting any ASCII character. Try program listing 1 to see how it works. Then change the two GOSUB 5 statements to GOSUB 1000 calls, and see the difference in operation. It's for precisely this reason that line 5 was included. Normally you wouldn't include the G3=1 before a GOSUB 5, but its included in the listing to accommodate the change to GOSUB 1000.

A GOSUB 5 is not intended for one character string input. Use GOSUB 1000 for this, because that allows on-the-spot editing of the input character.

GOSUB 1000 is best for input and editing of alphanumeric strings, but since your programs shouldn't require going through a menu except for major rerouting, you must access prompts directly from the input screen.

To use the label line for your prompts, you should insert a CALL 16959 (disable scroll) in your program's initialization line and a CALL 16964 (enable scroll) in your program's exit line. These calls will allow unrestricted use of the label line so that you don't need to worry about your headings being scrolled off the screen.

Answering the Prompts

Answering your label line prompts requires that you remember only one thing: Always return to the first position in the field to answer. If your input string consists of just one character, it will be checked as a prompt, but will be returned in K as your input too. (Be careful here. I suggest that you get used to using only alphabetic characters for prompting to avoid any future clashes should you answer a query with a single digit integer value while one of the prompts is looking for the same thing. If this should happen, prompt will win, intercepting and running off with your answer.)

You can press Enter from anyplace in the field to enter your input string, but prompts are accepted only from the first field position. Why? The routine watches for three simultaneous conditions:

- * The ENTER key has been pressed.
- * The cursor position equals field position number two (where it would be after entering a character in position one).
- * The input string is longer than one character.

Should all of these conditions be met, the first character of the string is truncated and used for relational comparison. The rest of the screen will be returned in K as the input. Should you inadvertently press Enter while in field position two, your returned string will be missing its first character. You'll have to reenter the routine type in the missing character, move the cursor back, forward or down with an arrow key and press Enter.

The Right Answer

When juggling numeric data, E0 will be returning your figures from the routine. If you're using these figures as monetary values, send them through the following line for rounding to two decimal places -- otherwise, your extended values will be incorrect.

```
NEW FIGURE = INT(OLD  
FIGURE*100+.5)/100:RETURN
```

When sending it to the routine for editing, first set your numeric value into E0 and then through the following line before calling GOSUB 1000:

```
N=STR$(E0):N=MID$(N,2+(E0<0),  
(LEN(N)+(E0>-1))):RETURN
```

Its purpose is to remove the leading or trailing blanks which BASIC inserts during string conversion, necessary for proper screen presentation. Notice that if your value is negative, the routine will remove only the trailing blanks, and will leave the minus sign alone.

You must also change the ON ERROR GOTO 556 statement to match your own program's error-trapping routine line. If you don't use one then insert a line somewhere that contains only RESUME and send it there.

While typing in the routine, don't let the apparent errors bother you. They ARE errors -- used intentionally to save coding space.

Well, that's it. I hope you enjoy the routine; I have found it to be to my programs what the microwave was to my kitchen: the missing link.

MISLIN.BA, the general-purpose input I/O subroutine.

```
5 K=INPUT$(1)  
6 O0=(K="D" OR K="d")  
7 O1=(K="E" OR K="e")  
8 O3=(K="Y" OR K="y")
```

```
9 O4=(K="N" OR K="n")  
10 O5=(K="F" OR K="f")  
11 O6=O3+O4+1  
12 O7=(K="B" OR K="b")  
13 G0=(K="C" OR K="c")  
14 O2=LEN(K)  
15 O8=(VAL(K)<=0)  
16 O9=(K="")  
17 RETURN  
1000 IF G3<1 OR G3>239 THEN BEEP  
      :K=""  
      :GOTO 6 ELSE O1=G2+G3  
      :K0=""  
      :K=N  
      :G=LEN(K)  
      :O=G3-G  
      :G4=G2  
      :ON ERROR GOTO 1390  
      :PRINT @G1,F  
      :PRINT @G2,N" "  
      :IF O THEN PRINT  
        @G2+G,STRING$(O,"_")" "  
1010 PRINT @G1,FQ  
1020 PRINT @G4,;  
1030 K0=INPUT$(1)  
1040 O=ASC(K0)  
1050 O4=G2+G  
1060 O5=G4-G2  
1070 O6=(G4>O4)  
1080 O7=O4-G4  
1090 O8=(G4<O4)  
1100 G5=G-O5  
1110 F0=LEFT$(K,O5)  
1120 IF O=13 THEN ON ERROR GOTO 556  
      :IF G THEN O=ASC(LEFT$(K,1))  
      :IF G4=G2+1 AND G>1 THEN  
        K0=RIGHT$(K,G-1)  
      :K=LEFT$(K,1)  
      :GOSUB 6  
      :K=K0  
      :GOTO 14 ELSE 6 ELSE 6  
1130 PRINT @G4,CHR$(27)"Q"  
1140 IF O>31 AND O<123 THEN 1320  
      ELSE IF O=2 OR O=20 THEN ELSE  
      IF O=8 THEN 1250 ELSE IF O=9  
      THEN 1410 ELSE IF O=17 OR  
      O=18 THEN 1180 ELSE IF O<27  
      THEN 1160 ELSE IF O=27 THEN  
      1380 ELSE IF O<32 THEN 1220  
      ELSE IF O=127 THEN 1240 ELSE  
      BEEP  
      :GOTO 1010
```



```

1150 IF O=20 THEN G4=G4 MOD 40+G2
:GOTO 1010 ELSE G4=G4 MOD
40+((O4\40)*40)
:GOTO 1010
1160 IF O=23 THEN G4=G2 ELSE IF
O=26 THEN ! ELSE
O3=INSTR(RIGHT$(K,O7)," ")
:IF O=6 THEN IF O3 THEN
G4=G4+O3 ELSE ! ELSE IF O1
THEN FOR O2=1 TO 26
:O3=INSTR(RIGHT$(LEFT$(K,O5-1)
,O2)," ")
:IF O3 AND O3<O5-1 THEN G4=G4-O2
:O2=26
:NEXT ELSE NEXT
:G4=G2
1170 GOTO 1010
1180 O3=(G4\40)*40
1190 O2=O3+39
1200 IF O=18 THEN IF O4>O2 THEN
G4=O2 ELSE ! ELSE IF O=17
THEN IF G2=>O3 THEN G4=G2
ELSE G4=O3
1210 GOTO 1010
1220 PRINT @G4,CHR$(27)"P"
1230 IF O<30 THEN G4=G4+((O=29 AND
G4>G2)-(O=28 AND G4<O1))
:GOTO 1010 ELSE G4=G4+(((O=30
AND G2<G4-39)-(O=31 AND
O1>G4+40))*40)
:GOTO 1010
1240 IF G3=1 THEN 1270 ELSE IF
G4=>G2 AND O8 THEN 1280 ELSE !
1250 IF G4>G2 AND O8 AND G3>1 THEN
G4=G4-1
:GOTO 1280
1260 IF G4=G2 THEN IF G THEN 1010
ELSE K=""
:G=0
:GOTO 1010 ELSE IF G4<=O4 AND
G4=>O1 THEN K0="" ELSE IF
G4=O4 THEN K0="" ELSE IF
G4<=O1 THEN ! ELSE 1010
1270 IF G3=1 THEN K=""
:G=0
:G4=G2
:PRINT @G4,"_"
:GOTO 1010 ELSE G4=G4-1
:G=G-1
:K=LEFT$(K,G)
:PRINT @G4,K0
:GOTO 1010

```

```

1280 PRINT @G4,CHR$(27)"P"
1290 G=G-1
1300 K=LEFT$(K,O5+(O=8))+RIGHT$(K,G
5+(O=127))
1310 IF O4<=O1 THEN PRINT @G2,K"_"
:GOTO 1010 ELSE PRINT @G2,K"_"
:GOTO 1010
1320 IF G4=G2 AND O4=O1 THEN K=K0+K
:PRINT @G2,K
:GOTO 1350 ELSE IF O4<O1 THEN
IF O8 THEN K=F0+K0+RIGHT$(K,G5)
:PRINT @G2,K
:GOTO 1350 ELSE IF G4<O1 THEN
IF O6 THEN G4=O4 ELSE ELSE
ELSE BEEP
:GOTO 1010
1330 PRINT @G4,K0
1340 K=K+K0
1350 G4=G4+1
1360 G=G+1
1370 GOTO 1010
1380 IF O6 THEN ! ELSE PRINT
@G4,STRING$(O7,"_")
:K=F0
:G=LEN(K)
:GOTO 1010
1390 G4=O4
1400 RESUME 1010
1410 IF O4+5>O1 THEN O3=O1-O4
:BEEP ELSE O3=4
:IF O6 THEN G4=O4
1420 K=F0+SPACE$(O3)+RIGHT$(K,G-(G4
-G2))
1430 PRINT @G2,K
1440 G4=G4+O3
1450 G=G+O3
1460 GOTO 1010

```

KEEPING IT CLEAN

When we speak of protecting a computer, we're usually referring to data security or surge protection. But that's not the only type of protection needed: A computer is a sensitive piece of electronic equipment and its physical environment is crucial.

Computers have come long way from the days of air-conditioned glass rooms and lab coats, and the Model 100 and Tandy 200 are

ruggedly constructed. Still, computer programs can be destroyed as the result of failure to provide a safe working or storage environment for the computer and peripherals.

In business settings, anti-static mats and somewhat clean desks are to be expected -- but in the field, who knows what the laptop might encounter? At an airport lounge or in the Gobi desert, the worry of protecting data isn't as important as protecting the keyboard and screen. Smoke, dust, soot, aerosol sprays, carpet fibers, pet hair, airborne cooking grease, bits of food and particles of printer paper are just a few of the many pollutants that can cause serious internal problems.

Here are a dozen suggestions that will help keep your portable -- or any computer -- in top working condition:

1. Don't allow any eating, drinking or smoking in the vicinity of the computer -- yes, that includes you. Ever clean coffee stains or cigarette ashes out of a keyboard? It's not a pleasant task.
2. When removing disk and cassettes, put them right into their storage boxes or envelopes and put them into permanent storage.
3. Don't place disks or tapes on top of an external monitor, near the recharger or disk drive. The heat could warp the disk and the electromagnetic field could scramble your data.
4. Don't place disks near a telephone. When the phone rings, your data could be lost. Florescent desk lamps, high intensity desk lamps, radios and speakers all contain electromagnets or permanent magnets which could cause erasure of magnetically stored data.
5. Don't use a pencil to fill in data labels on diskettes. Graphite is a conductor and even the smallest particle on a disk can cause problems. Use a felt-tip marker for annotating disk labels.
6. Don't use a portable kerosene heater in the same room with a computer. The kerosene mist that permeates the room will leave a conductive film of the disk.
7. Don't place your computer in the same room with a fireplace or a wood stove. The ash, smoke and dust will contaminate the system.
8. Don't ever block cooling air outlets. To do so will cause heat buildup which can result in damaged components and costly repairs.
9. Don't allow direct sunlight to strike your computer. Install a reflective window film or draw the drapes to prevent excessive heat which will shorten the life of circuit chips and other components. Also, be careful when leaving the computer or disks in the glove compartment or on the seat of your car during below-freezing weather or on a sunny day.
10. Use a damp (not wet) cleaning cloth to clean dust and residue from system components.
11. To clean in cracks, crevices or other inaccessible areas with a damp cloth, use a canister of pressurized clear air and tweezers.
12. Keep your computer and printer covered at all times when not in use -- preferably in a drawer, briefcase or cabinet.

INTERNAL AUDIT

Many moons ago the Internal Revenue Service decided to buy a few laptop computers for its field employees. Okay, more than a few: 17,000 units, with a potential price tag of \$36 million.

Many major portable computer manufacturers were involved in the bidding process, including Grid, Hewlett-Packard, Data General and IBM. IBM? They don't have a battery-powered computer.

That wasn't important. Since one of the stipulations of the IRS contract was that the computer be commercially available, Big Blue would presumably introduce the computer a few days before the contract was awarded, and walk away with 17,000 orders.

That was everyone's expectation: several of the bidders threatened lawsuits, and the prestigious industry newspaper PC Week even showed sketches of the new, but still unannounced, IBM portable computer. At the last minute the IRS decided to delay awarding the contract, so we don't know what will happen.

Despite the potential lawsuits, many manufacturers of high-powered portables would welcome an IBM portable: Although they're bound to lose business to IBM, they hope that Big Blue will "legitimatize the market" for all portables.

Not Tandy. Tandy is dominating the market with its three portable computers -- computers that perform functions that owners demand, not merely demonstrate that new technology can run from batteries. And in the long run, Model 100 and Tandy 200 owners will win -- with or without IBM's legitimatization.

Alan L. Zeichick
Editor

PORTABLE PROGRAM REVIEW
P.O. Box 250
Highland Mill
Camden, ME 04843

FIRST CLASS U.S. POSTAGE PAID CAMDEN, ME PERMIT NO. 5

SUBSCRIBER: PLEASE ADVISE IF ABOVE ADDRESS IS INCORRECT

PORTABLE PROGRAM REVIEW is published monthly and copyright © 1986, all rights reserved, by Camden Communications Inc., a Maine corporation with offices at Highland Mill (P.O. Box 250), Camden, Maine 04843. Contents of this publication may not be reproduced in whole or in part unless expressly authorized in writing by the publisher. Tandy and Radio Shack are registered trademarks of Tandy Corporation. U.S. Subscription \$29.97. Please address all subscriptions and inquiries to Nancy Wight, Circulation Director, (207) 236-4365. ISSN: 0883-7295.